A TILE-BASED APPROACH FOR PHOTO-REALISTIC VOLUME RENDERING

by

MANISH MATHAI

A THESIS

Presented to the Department of Computer and Infomation Science
and the Graduate School of the University of Oregon
in partial fulfillment of the requirements
for the degree of
Master of Science

June 2018

THESIS APPROVAL PAGE

Student: Manish Mathai

Title: A Tile-Based Approach for Photo-Realistic Volume Rendering

This thesis has been accepted and approved in partial fulfillment of the requirements for the Master of Science degree in the Department of Computer and Infomation Science by:

Hank Childs	Chair

and

Sara D. Hodges	Interim Vice Provost and Dean of the Graduate School

Original approval signatures are on file with the University of Oregon Graduate School.

Degree awarded June 2018

THESIS ABSTRACT

Manish Mathai

Master of Science

Department of Computer and Infomation Science

June 2018

Title: A Tile-Based Approach for Photo-Realistic Volume Rendering

Previous studies on photo-realistic volume rendering have failed to optimize for performance with respect to the cache-hierarchy. With this thesis, we consider a tile-based approach for photo-realistic volume rendering, in an effort to improve cache performance and decrease overall execution time. We evaluated the algorithm compared to the traditional approach, with workloads of varying data sizes, resolutions, samples per pixel. Overall we ran 48 serial experiments, and found that the tile-based approach is consistently faster than the traditional approach, including speedups of up to 20%. Additionally we determine that the improvement does not carry forward directly to parallel platforms like Intel TBB.

CURRICULUM VITAE

NAME OF AUTHOR: Manish Mathai

GRADUATE AND UNDERGRADUATE SCHOOLS ATTENDED:

University of Oregon, Eugene, OR, USA
University of Mumbai, Mumbai, MH, India

DEGREES AWARDED:

Master of Science, Computer and Information Science, 2018, University of Oregon
Bachelor of Science, Information Technology, 2011, University of Mumbai

AREAS OF SPECIAL INTEREST:

Photo-realistic Rendering
GPU Programming
High Performance Computing

PROFESSIONAL EXPERIENCE:

Research Assistant, Computing and Data Understanding at Extreme Scale, University of Oregon, 2017-2018
Computation Intern, Lawrence Livermore Nation Lab, 2017
Teaching Assistant, Fluency with Information Technology, 2017
Software Engineering, Outernet Inc, 2015-2016
Game Programming, Rolocule Games, 2012-2015
Technical Analyst, Morgan Stanley, 2011-2012

GRANTS, AWARDS AND HONORS:

J. Donald Hubbard Family Scholarship, University of Oregon, 2017

PUBLICATIONS:

Lessley, B., Perciano T., Mathai M., Childs H., & Bethel. E. W. (2017). Maximal clique enumeration with data-parallel primitives. *2017 IEEE 7th Symposium on Large Data Analysis and Visualization (LDAV)*, 16-25

ACKNOWLEDGEMENTS

I would like to thank my advisor Hank Childs, whose support was instrumental in completing this work. I would also like to thank the entire CDUX group for their inspiration and love. Finally, I would like to thank my family for their support.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER I

INTRODUCTION

Volume rendering is an important visualization tool in many fields of scientific study including medical imaging, non-destructive material inspection, fluid simulation, etc. It enables visual understanding and exploration of scalar fields defined over volumetric data, using 3D graphical rendering techniques. The technique involves sampling the scalar field to render an image, and does not generate intermediary surfaces or geometric primitives. The rendered image represents the 3D volume as a whole and lets the viewer inspect the interiors of the volume through use of appropriate transfer functions.

Volume rendering can achieve greater depth perception and enhancement of internal structures by using a physically-based rendering model, which adds photo-realism to the rendered volume. In such a model, the volume is considered to be an inhomogeneous participating media which emits, absorbs and scatters light as it transported through it. This resulting realistic illumination and shadows has been shown by Lindemann and Ropinski (2011) and Ropinski, Dring and Rezk-Salama (2010) to improve the correctness and swiftness of professional analysis of the scientific volume data.

In recent years, physically-based rendering model has been one of the more dominant techniques for augmenting realism in generated images. This model mimics the physical behavior of light, at micro and macro levels, using accurate approximations of the rendering equation, as noted by Kajiya (1986). In practice this involved simulating the path of many rays of light through the volume, also called the participating media. These light rays, as they pass through the volume, are subject to physically plausible events like absorption and scattering. These events are governed by the properties of the volumetric data and the simulated light sources. Since the model attempts to reflect reality, it is able to reproduce many real world phenomenon like soft shadows, ambient occlusion, depth of field, color bleeding, etc. without having to be explicitly implemented.

The rendering equation, while being a powerful technique, is also intractable to solve analytically as it is a recursive integral. In practice, approximation methods like Monte Carlo are used. Monte Carlo-based light transport has the added benefit of being unbiased and thus retains physically accurateness of the image.

Volumetric data sets tend to be large and often need to be loaded completely into memory, irrespective of how much of the volume is visible or contribute to the image due to the transfer function used. Due to the stochastic nature of Monte Carlo methods, while tracing adjacent light rays originating from the close points, they may travel through different parts of the volume and hence lead to cache contention and slower performance.

Ray tracing through a volume is characterized by a highly view-dependent data access pattern. The typical cache hierarchy of modern CPUs uses L1, L2, and L3 caches and clearly cannot store the entire volume data in memory and the data access patterns involved in ray tracing can cause significant cache thrashing. Additionally, Monte Carlo methods require multiple estimate images to be generated, requiring several passes and hence multiple loads of the volume through the cache hierarchy.

With this thesis we introduce an improvement to photo-realistic volume rendering which boosts the runtime of the Monte Carlo ray tracing algorithm by upwards of 20%. In order to improve cache utilization and reduce cache thrashing, we split the estimates generated into tiles of sizes much smaller than the final image and render the estimates for each tile successively. This has the effect of drastically reducing the required volume data and thus decreases the probability of eviction.

CHAPTER II

RELATED WORK

2.1 Foundations of Volume Rendering

Volume rendering can be achieved using image-order algorithms, which iterate over the pixels, or using object-order algorithms, which iterate over the volume elements. In this section, we discuss three major flavors of these algorithms and some noteworthy extensions.

2.1.1 Ray Casting. Drebin, Carpenter and Hanrahan (1988) presented an image-order technique for volumes consisting of a mixture of different materials, for example bone, soft tissue and fat, with varying optical properties. The authors used classification techniques, which are either user-provided or based on probabilistic models, to determine the contribution of all the materials in each volume element (voxel) as percentages. Once the voxel material compositions are calculated, transfer functions are then used to map from material classifications to corresponding optical properties like RGB colors and opacity, weighted by the calculated percentages. These colors and opacity are summed up as the final voxel color.

The final image is generated by casting rays from the viewer's position through each pixel, traversing the volume. As the rays pass from one voxel to the next, surfaces between materials are detected using changes in the values between the voxels and are used to simulate light scattering. At points of scatter, the gradient of the neighboring voxels is used as the surface normal for diffuse shading calculations. The resulting color is a function of the surface normal, voxel color and the color of the light source.

A significant addition to this algorithm was done by Mueller, Moller and Crawfis (1999), where the voxel values were interpolated along the ray first, and then the transfer functions were applied after the scatter point was determined. This modification preserved higher frequency details in the voxels, which were lost in the previous technique, and thus greatly reduced blurring in the final image.

2.1.2 Splatting.

In contrast to the ray tracing approach, Westover (1990) proposed an object-order rendering algorithm, which splats the voxels into the image, using 3D Gaussian kernels. The kernels are weighted by voxel values, creating 2D "footprints" of the voxels. The footprints are then composited in back-to-front (or front-to-back) order. The algorithm has the advantage of reduced memory footprint (via elimination of empty voxels), but it limits the use of optimization techniques like early ray termination or adding realistic effects like glossy reflections and soft shadows.

2.1.3 Shear-Warp Factorization.

Lacroute and Levoy (1994) introduced another object-order rendering algorithm called shear-warp factorization, where the volume slices are transformed and resampled into an object space that aligns it parallel to the image plane. This enables the algorithm to render the volume using a scanline-based rasterization technique. An intermediate but distorted image is generated, that is then warped into the final image by applying a reverse of the original transformation. The authors presented variants of their algorithm to accommodate parallel and perspective projections.

2.2 Photo-realism

Perception of relative importance of regions and spatial relations are accentuated by global illumination and shadows, as noted in Wanger, Ferwerda and Greenberg (1992). The authors demonstrated significant increase for users, with respect to accuracy of position, scaling and orientation detection tasks, by use of correct shadowing cues in generated images.

High quality shadows were introduced by Lokovic and Veach (2000), for both mesh and volumetric data, through the use of pre-filtered deep shadow maps which exhibit faster lookups. Hadwiger, Kratz, Sigg and Bühler (2006) adopted the algorithm to volumetric ray casting on GPUs, generating anti-aliased images using a pre-computed lookup table.

Zhukov, Iones and Kronin (1998) introduced approximations for global illumination through obscurances, by determining the occlusion of ambient light at interpolated points. However, their technique does not account for indirect illumination and only considers local geometry and thus is not physically accurate. Our algorithm, as a direct consequence of the shadowing computation, considers the entire volume for occlusion detection.

2.3 Cache Thrashing

3D volumes are often large, laid out in physical memory sequentially, and are moved up through the cache hierarchy in a linear fashion. In order to counteract the strong view-dependence of data access patterns, Parsonson, Bai, Bourn, Bajwa and Grimm (2011) proposed subdivision and reorganization of the data into small contiguous blocks. Their technique had the advantage of reducing the total memory used, combining empty blocks into larger ones and increasing cache friendliness. That said, their work required reorganizing internal data structures, which may not be possible with in situ processing. Hence, we do not consider their approach as a feasible optimization for this work. On the other hand, our work can be adopted for in situ rendering of volumetric data, without significantly impacting simulation codes.

Casting rays through a volume requires multiple accesses per voxel. This leads to increased main memory accesses, which slows performance. Levoy (1990) suggested techniques like early ray termination and coherence encoding as ways to reduce the total number of memory accesses. These techniques, however, increase the complexity of the rendering pipeline, as they require complex pyramidal structures to accommodate resampling.

Knittel (2000) proposed a comprehensive system called ULTRAVIS, with many optimizations, including cache-based optimizations, specifically for Pentium III CPUs. The system describes a special layout for volume data in main memory, which causes frequently used data to be "locked" onto cache lines, available for fast access. However, the approach requires extensive re-arrangement of volume data and reduces overall cache capacity. Parker, Shirley, Livnat, Hansen and Sloan (1998) proposed a similar technique of organizing the voxels into "bricks" in memory, for interactive isosurface rendering. Parker et al. (2005) used the same technique for interactive volume rendering.

2.4 Photo-realistic Volume Rendering

Max (1995) reviewed several optical models for light transport through volumes consisting of materials, including the single scattering model used in our algorithm.

The author presented the equations for the bidirectional reflection distribution function (BRDF) and probability density function by modeling the volume as small spherical particles. Two works, by Cook, Porter and Carpenter (1984) and by Kajiya (1986), presented the use of Monte Carlo-based estimation methods for solving shading integrals numerically. Two additional works, by Rushmeier (1988) and by Csebfalvi and Szirmay-Kalos (2003), applied similar methods to volume rendering, using random sampling of the volume using its probability density function.

Schlegel, Makhinya and Pajarola (2011) presented an optimized GPU-based solution, with features like ambient occlusion, color bleeding and soft-shadows. Our work contrasts with theirs, in that we focus on CPU architectures and the cache benefits from tiling.

2.5 VTK-m

Our algorithm was implemented using VTK-m introduced by Moreland et al. (2016), which is a library for many-core visualization that uses data parallel primitives as building blocks. With this library, algorithm development does not consist of C++ programs with while or for loops, but rather using primitives such as map, reduce, gather, scatter, etc. As a result, mapping a new algorithm to the paradigm requires re-thinking the algorithm from the perspective of data parallel primitives (and not just porting). Several previous works have investigated individual algorithms. Examples include: Carr, Weber, Sewell and Ahrens (2016) with contour tree computation, Larsen, Meredith, Navrtil and Childs (2015) with ray-tracing and Larsen, Labasan, Navrátil, Meredith and Childs (2015) for unstructured volume rendering, Lessley, Binyahib, Maynard and Childs (2016) with external facelist calculation, Lessley, Moreland, Larsen and Childs (2017) with hash performance, Lessley, Perciano, Mathai, Childs and Bethel (2017) and with maximal clique calculation, Li et al. (2017) with wavelet compression, Lo, Sewell and Ahrens (2012) with isosurface generation, Maynard, Moreland, Atyachit, Geveci and Ma (2013) with thresholding, Schroots and Ma (2015) with cell-projected volume rendering, and Widanagamaachchi et al. (2014) for connected component calculation. Our work contrasts from these previous works since we consider a new algorithm (physically-based rendering).

CHAPTER III
ALGORITHM

In this chapter, we describe our algorithm for generating high quality photo-realistic volume visualizations. It resembles the one described by Kroes, Post and Botha (2012), with the key difference being our use of tiling to increase cache locality and thus decrease the cache thrashing. We first discuss the details involved in the physically-based shading model we use and the realistic effects it simulates and then discuss our tile-based algorithm.

3.1 Background on Physically-Based Rendering

At a high level, physically-based rendering works by estimating the total light contribution arriving at each pixel of the image. The light contributions are computed iteratively, with each iteration generating an approximation of the incident light, called frame estimates. These approximations are then combined together using Monte Carlo integration into the final image.

Physically-based rendering algorithms solve the rendering equation, which models the light (irradiance) leaving from any given point, as introduced by Kajiya (1986) and by Immel, Cohen and Greenberg (1986).

$$L_o(x,v) = L_e(x,v) \ + \int_{\Omega} f_r(x,v',v)\ L_i(x,v')\ (v' \cdot n)\ dv'$$

In the above equation the outgoing light (L_o) at point x in the viewing direction v is computed as the sum of the light emitted (L_e) by x and reflection of the incoming light (L_i) integrated over the unit hemisphere Ω containing all possible incoming directions v'. f_r is the bidirectional reflectance distribution function (BRDF), controlling the proportion of light reflected from v to v'. $(v' \cdot n)$ factors in the attenuation due to angle of incidence, from Lambert's cosine law, where n is the surface normal.

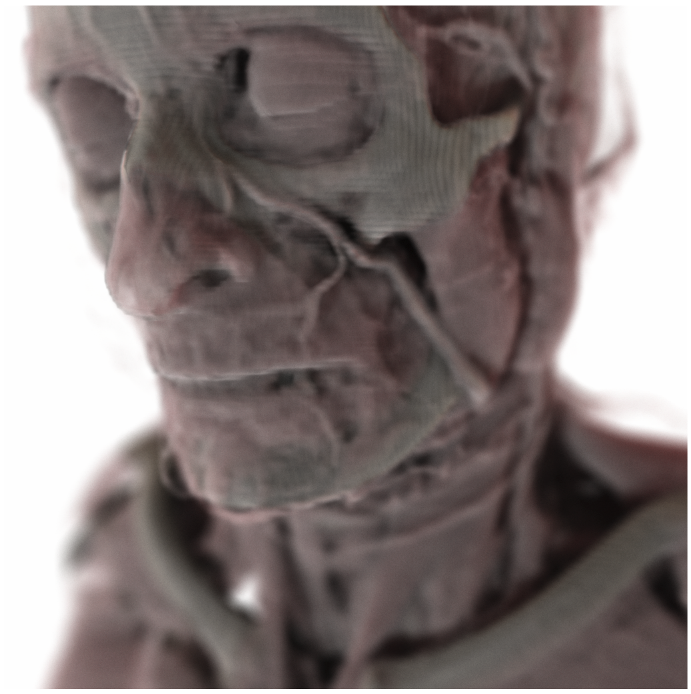

Figure 1. Rendering of Manix dataset, after 2000 iterations, with soft shadows, depth-of-field and self shadowing.

The integral involved in the rendering equation clearly indicates its recursive form, making it impervious to analytic solutions for the general case. Instead Monte Carlo integration can be used to solve the integral term, by converting it into a finite sum. With each iteration, an increase in the number of samples used in the sum reduces the variance while converging to the true value. As noted by Newman and Barkema (1999), after N iterations of generating stochastic estimates for each pixel, Monte Carlo integration, augmented with importance sampling, calculates the expected value C as

$$C = \frac{1}{N} \sum_{i=1}^{N} \frac{f(X_i)}{p(X_i)}$$

3.2 Algorithm Overview

We describe our algorithm in five parts: camera ray generation (3.2.1), light transport (3.2.2), shading calculation (3.2.3), tiling (3.2.4) and other details (3.2.5).

3.2.1 Ray Generation. In our algorithm, camera rays are generated in a discrete step, once per iteration. For an image with pixel width W and height H, W x H rays are traced through the volume per iteration.

We model a virtual camera using a thin lens model described by Barsky, Horn, Klein, Pang and Yu (2003), which ignores the optical effects caused due to the thickness of the lens. The model incorporates the depth-of-field effect, i.e., circle of confusion, by simulating an aperture of a user-defined radius, which is demonstrated in Figure 1. The camera rays are described using the parametric equation $P(t) = \vec{O} + t * \vec{D}$. The origin $\vec{O}$ of each ray is selected as the camera position, jittered by adding a stratified sampled point on a disk with the radius of the aperture. The direction $\vec{D}$ is calculated by randomly sampling a point within the area of the corresponding pixel.

3.2.2 Light Transport. One of the challenges of volume rendering involves estimating scattering events for the camera rays due to the absence of explicit surface geometry. Additionally, the heterogeneous nature of the volume requires consideration of multiple sample positions along the ray. Our algorithm simulates one single scattering event for each ray that intersects with the volume.

For each ray, an intersection test is performed against the bounding box of the volume, yielding an entry point and an exit point. The rays which fail the test are intersected against the background light and their light contribution is calculated accordingly. The rays which pass the bounding box intersection test are then evaluated by applying the classic ray marching algorithm through the volume, with a fixed step size. The ray marching continues until a point in the volume is reached where the probability of the ray being scattered by the volume is sufficiently high. This point is designated as the scattering point for the ray.

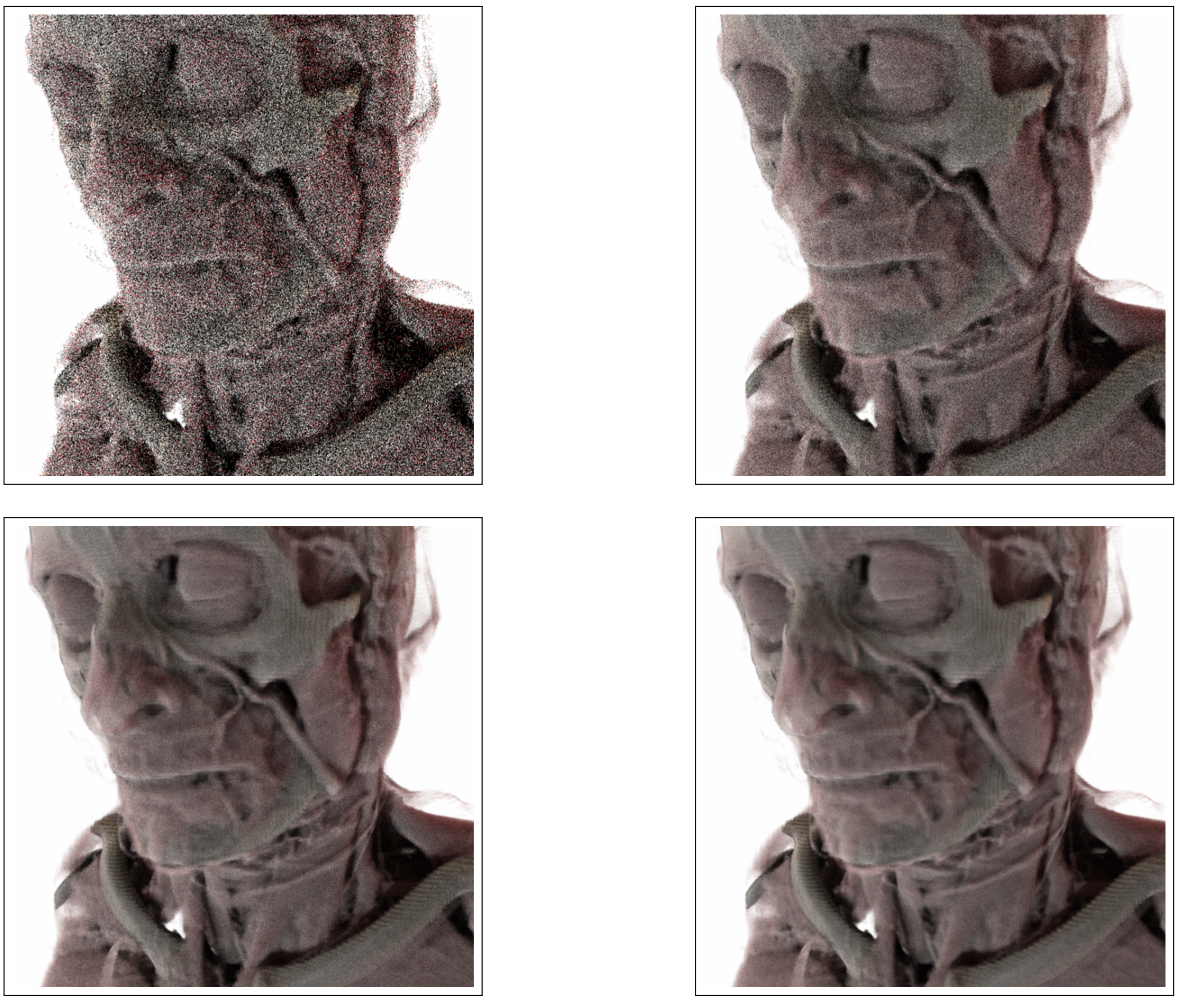

Figure 2. Manix dataset rendered with 1, 10, 50, and 100 iterations, from top to bottom, left to right.

In our algorithm, we model only one scattering event for the lifetime of a ray. The scattering point is computed as the point along the ray direction, where the cumulative extinction coefficient σ_t carried by the ray exceeds the maximum extinction coefficient σ_{max}, as described in Raab, Seibert and Keller (2008).

3.2.3 Shading. While our algorithm can support use of any number of arbitrarily shaped lights, the implementation supports multiple area lights and one background light. The lighting contribution at each scattering point is estimated by calculating the following two components:

1. The contribution from direct lighting with attenuation is calculated using next event estimation, by connecting the scattering point with a randomly chosen point on a randomly chosen light.

2. The contribution from indirect lighting is calculated by scattering the ray in the direction generated by sampling the BRDF.

We ensure that the two forms of sampling are weighted accurately by using multiple importance sampling with the power heuristic.

3.2.4 Tiling. In our algorithm, the following type of data needs to be frequently accessed as the image estimates are generated:

– Volume data, with each voxel using 8 data points.

– Four transfer functions, which map from the volume value to roughness, and diffuse, specular and emission colors.

– Ray parameters, with 4 floating point values.

– Additional parameters for shading calculation, like lighting.

– Color buffers used for image estimate and final image.

The physically-based rendering approach requires an image to be rendered repeatedly, with ray jitter on each successive iteration. The traditional approach of tracing rays in the scanline order, row by row, causes the first three types of data from the above list to be frequently loaded and evicted from cache lines, leading to inefficient cache usage. This cache thrashing leaves a significant amount of performance on the table. Our tile-based algorithm is designed to improve cache efficiency of a physically-based rendering model, where multiple rays are cast through the same pixel across many iterations. Our algorithm resolves the problem by the use of tiling, where we divide the image plane into sub-parts and trace rays through each tile one at a time and collect all the frame estimates for the tile at one go. Tiling is based on the idea that the voxels and data accessed during the tracing of one ray can be reused for other rays which are likely to be close to the first one. By splitting the image plane into tiles of closely related pixels, we limit the amount of data accessed for calculating the frame estimate of the tile. Further, as all the frame estimates of a single tile are computed one after other, the data loaded into the cache is less likely to have been be replaced with other data. This will reduce the time spent by the CPU waiting for main memory accesses, improving the total render time. The magnitude of benefit is a key evaluation done in this thesis.

3.2.5 Other Implementation Details. The algorithm is designed to operate on volumes which are defined in terms of 3D grids. The data is available only at the grid points, i.e. discrete points spread regularly across the volume. However, our algorithm requires the ability to sample the volume data at any arbitrary point in the volume. We implement trilinear interpolation in order to compute the volume data at the sample point, using the values available at the eight corners of the cell containing it. In effect, this converts the discrete input data into a continuous field defined over the entire 3D volume.

Our algorithm requires sampling from uniform random distribution at various places. Typical implementations for random number generators (RNG) require few bytes of state to be stored between successive invocations. However, this shared state property requires synchronization in order to be used between multiple threads of execution, which in turn can cause a bottleneck. In order to avoid this, we use one instance of 64-bit variant Xorshift RNG per pixel, storing the shared state in a separate buffer. Since we do not share the state across pixels, this allows the algorithm to be parallelized over pixels.

CHAPTER IV

EXPERIMENT OVERVIEW

4.1 Implementation

The focus of our experiments is to study the performance impact of using a tile-based approach for physically-based volume rendering. The code for this experiment was implemented using the VTK-m framework as described by Moreland et al. (2016). VTK-m supports compilation and generation of optimized code for multiple backends including the CPU and GPU. For the CPU, it supports running in a serial mode as well as in a parallel mode, using Intel Thread Building Blocks (TBB) as the parallelization mechanism. We conducted out study in two phases: serial and parallel.

4.2 Configurations

For the serial phase, our study varied three factors:

– Data Set: 3 options

– Camera Position (zoomed in and zoomed out): 2 options

– Tile Size: 8 options

We ran our experiment on the cross product of these options, resulting in a total of $3 \times 2 \times 8 = 48$ tests. Each of the tests generated an image of resolution 1024 $\times$ 1024. Finally, we selected one transfer function per data set as it did not significantly impact results.

For the parallel phase, we limited our study to the four largest tile sizes, as TBB did not parallelize over the smaller tile sizes. We used the same data sets and camera positions as the serial phase, leading to a total of $3 \times 2 \times 4 = 24$ tests.

4.3 Hardware Architecture

Our experiments ran on an Intel Xeon E5. It contained 6 cores (12 logical cores) running at 3.5GHz.

4.4 Data Set

This study used the following volumetric data sets:

- Engine: An engine block with resolution 128 × 128 × 64.
- Manix: CT scan of a human head with resolution 256 × 230 × 256.
- Macoessix: CT scan of a human pelvis with resolution 512 × 461 × 512.

4.5 Camera Position

It is generally understood that one of the dominant factors in ray tracing is ray-volume intersection. In order to examine the effect of varying the number of successful ray-volume intersections, we choose two camera positions for each data set. One of the camera positions was at a close position, relative to the center of the data set, and the other was further out.

4.6 Tile Size

The tile sizes were selected as powers of two so as to ease the division of the image into appropriate tiles. The following tile sizes were examined for the serial phase: 2, 4, 8, 16, 32, 64, 128, 256. For example, the tile size of "2" corresponds to rendering in 2 × 2 pixel tiles, meaning that 512 × 512 (or 260k) total tiles were considered.

For the parallel phase, the following tile sizes were used: 64, 128, 256, 512.

4.7 Testing Procedure

For each test, ten renderings were performed. The first five renderings were used as warm-ups, and the average of latter five rounds were used for measurements. The process for generating each image involved rendering twenty individual estimates using tiling, which were continuously integrated into the final image by performing a cumulative moving average of the estimates.

4.8 Measurements

We measured the total execution time for rendering, i.e., the total time taken for rendering and integrating the twenty estimates, for both serial and parallel phases. I/O times we excluded from our study.

CHAPTER V

RESULTS

The results of the experiment are organized into three sections: serial phase (5.1), parallel phase (5.2) and generated images (5.3)

5.1 Serial Phase Results

In this section we examine the performance improvement of our tiling algorithm for all tests in the serial phase. For each data set, we compare the time taken to render an image using multiple tile sizes and compare it against the time taken when no tiling is used. We also examine the timing difference due to the various tile sizes. Note that for each of the timings listed, timings are measured in seconds and the leading tile size is indicated in **bold**.

5.1.1 Engine.

The engine data set is a comparatively small one and shows the fastest render time for both camera positions, amongst all the data sets. Table 1 shows a distinct trend of rendering times decreasing as the tile size reduces, with the lowest time shown by the smallest tile of size 2×2. This trend is visible for both close and far camera positions, showing a best-case reduction of render time by 3.6% and 11% over non-tiling, respectively. This decreasing trend is due

Size	Engine - Close	Engine - Far
Full	44.06	14.78
2×2	**42.47**	**13.16**
4×4	42.91	14.02
8×8	43.7	14.28
16×16	44.89	14.23
32×32	45.58	14.62
64×64	45.62	14.56
128×128	44.02	14.71
256×256	45.13	14.76
% reduction	3.61%	10.96%

Table 1. Serial phase timings for Engine, measured in seconds.

to the fact that as the tile size decreases, the total number of voxel and other data required per ray is limited. This helps accentuate the performance benefits of the increased cache locality.

5.1.2 Manix. The render timings for manix data set exhibit behavior similar to that of the engine data set, with the best case reductions being 20.4% and 7.4% for close and far camera positions respectively. An exception is seen for the far camera position, however, where the winning tile size is 4×4. The random nature of the light transport could be the likely explanation for this.

Size	Manix - Close	Manix - Far
Full	127.52	35.66
2×2	**101.51**	33.91
4×4	105.09	**33.02**
8×8	108.51	34.57
16×16	108.82	34.48
32×32	110.86	34.73
64×64	112.29	34.74
128×128	123.61	35.55
256×256	124.40	35.64
% reduction	20.40%	7.4%

Table 2. Serial phase timings for Manix, measured in seconds.

5.1.3 Macoessix. The macoessix data set shows the most significant reduction in rendering times: 22.8% and 14.3% for close and far camera positions, respectively. As the largest data set among all our experiments, macoessix reveals that the reduction in rendering times are likely to scale along with the dimensions of the volume.

These results clearly show that the tiling leads to a significant reduction in rendering runtime. Further, in all but one experiment, the tile size of 2×2 shows the largest reduction in runtime.

Size	Macoessix - Close	Macoessix - Far
Full	289.65	54.14
2×2	**223.58**	**46.38**
4×4	235.10	47.13
8×8	235.21	46.62
16×16	240.46	48.83
32×32	268.35	53.57
64×64	283.83	54.03
128×128	286.96	53.10
256×256	287.20	53.90
% reduction	22.81%	14.33%

Table 3. Serial phase timings for Macoessix, measured in seconds.

5.2 Parallel Phase Results

This section assesses the tiling-based approach in the context of the Intel TBB multi-threading environment. From Table 4, we observe that tiling with TBB produces a significant *increase* in the time required to render an image, as compared to the non-tiling baseline. While these results are non-intuitive, they can be explained by task scheduling mechanism used by TBB. In this TBB phase, we parallelize over the pixels of each tile, with multiple threads executing on ranges of those pixels across multiple cores. When smaller tiles are used, inter-thread cache contention is higher due to memory accesses of shared data. Higher tile sizes reduce the likelihood of multiple parallel threads accessing the same shared data, decreasing the cache contention and reducing the render timing. It should be noted that the TBB version of our algorithm exhibits lower runtimes for all equivalent tile sizes, as compared to the serial version.

Size	Engine - Close	Engine - Far	Manix - Close	Manix - Far	Macoessix - Close	Macoessix - Far
Full	5.46	1.69	15.21	4.56	35.49	6.95
64×64	26.52	8.25	54.88	17.16	108.26	23.21
128×128	7.78	2.48	20.99	6.13	49.14	9.94
256×256	6.03	1.92	16.41	5.03	38.51	7.89
512×512	5.63	1.81	15.38	4.62	36.06	7.19

Table 4. Parallel phase timings for rendering and integrating 20 estimates, measured in seconds.

5.3 Images

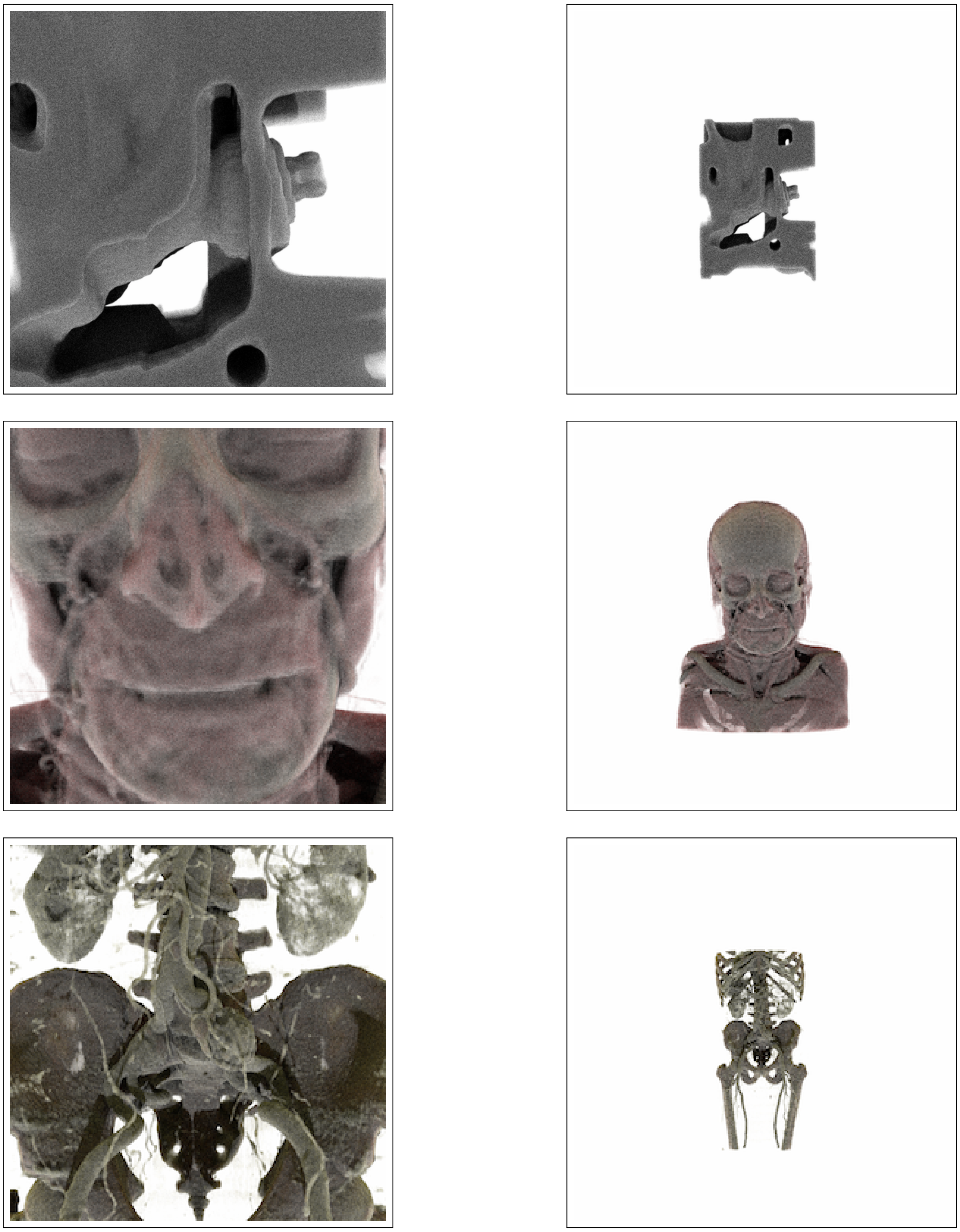

Figure 3. Close and far renders of Engine, Manix, and Macoessix

CHAPTER VI

CONCLUSION AND FUTURE WORK

We have described a tiling-based algorithm for physically-based volume rendering and performed a series of experiments that show that the tiling mechanism can bring about reduction of the rendering time when used in a serial manner. The process of generating multiple stochastic variants of the image responds positively, and exhibits a decrease in the total run time due to increase in cache locality and coherence introduced by our tiling algorithm. Finally, we find that the performance improvement displayed by the tiling approach does not directly transfer to shared-memory parallel platforms like Intel TBB.

In terms of future work, we would like to explore other enhancements that can potentially increase cache locality. One promising approach has been described by Larsen, Meredith et al. (2015), where the rays are re-ordered using a space filling curve. We would also like to investigate the effect of SIMD-based vectorization on the cache utilization and explore any potential modifications to the physically-based rendering algorithm. Additionally, we would like to further expand the tiling mechanism to take into account the scheduling peculiarities of parallel platforms such as TBB, as well as study the possibilities of a distributed tiling solution. GPU platforms have demonstrated success in using tiling-based rasterisation engines, especially in mobile devices. We would like to explore the portability of this tiling approach to server-grade GPUs, using nVidia's CUDA platform.

REFERENCES CITED

Barsky, B. A., Horn, D. R., Klein, S. A., Pang, J. A. & Yu, M. (2003). Camera models and optical systems used in computer graphics: part i, object-based techniques. In *International conference on computational science and its applications* (pp. 246–255).

Carr, H. A., Weber, G. H., Sewell, C. M. & Ahrens, J. P. (2016). Parallel peak pruning for scalable smp contour tree computation. In *Large data analysis and visualization (ldav), 2016 ieee 6th symposium on* (pp. 75–84).

Cook, R. L., Porter, T. & Carpenter, L. (1984). Distributed ray tracing. In *Proceedings of the 11th annual conference on computer graphics and interactive techniques* (pp. 137–145). New York, NY, USA: ACM. Retrieved from `http://doi.acm.org/10.1145/800031.808590` doi: 10.1145/800031.808590

Csebfalvi, B. & Szirmay-Kalos, S.-K. (2003). Monte carlo volume rendering. In *Proceedings of the 14th ieee visualization 2003 (vis'03)* (pp. 59–). Washington, DC, USA: IEEE Computer Society. Retrieved from `https://doi.org/10.1109/VIS.2003.10000` doi: 10.1109/VIS.2003.10000

Drebin, R. A., Carpenter, L. & Hanrahan, P. (1988). Volume rendering. In *Proceedings of the 15th annual conference on computer graphics and interactive techniques* (pp. 65–74). New York, NY, USA: ACM. Retrieved from `http://doi.acm.org/10.1145/54852.378484` doi: 10.1145/54852.378484

Hadwiger, M., Kratz, A., Sigg, C. & Bühler, K. (2006). Gpu-accelerated deep shadow maps for direct volume rendering. In *Proceedings of the 21st acm siggraph/eurographics symposium on graphics hardware* (pp. 49–52). New York, NY, USA: ACM. Retrieved from `http://doi.acm.org/10.1145/1283900.1283908` doi: 10.1145/1283900.1283908

Immel, D. S., Cohen, M. F. & Greenberg, D. P. (1986). A radiosity method for non-diffuse environments. In *Proceedings of the 13th annual conference on computer graphics and interactive techniques* (pp. 133–142). New York, NY, USA: ACM. Retrieved from `http://doi.acm.org/10.1145/15922.15901` doi: 10.1145/15922.15901

Kajiya, J. T. (1986). The rendering equation. In *Proceedings of the 13th annual conference on computer graphics and interactive techniques* (pp. 143–150). New York, NY, USA: ACM. Retrieved from `http://doi.acm.org/10.1145/15922.15902` doi: 10.1145/15922.15902

Knittel, G. (2000). The ultravis system. In *Proceedings of the 2000 ieee symposium on volume visualization* (pp. 71–79). New York, NY, USA: ACM. Retrieved from `http://doi.acm.org/10.1145/353888.353901` doi: 10.1145/353888.353901

Kroes, T., Post, F. H. & Botha, C. P. (2012). Exposure render: An interactive photo-realistic volume rendering framework. *PloS one*, *7*(7), e38586.

Lacroute, P. & Levoy, M. (1994). Fast volume rendering using a shear-warp factorization of the viewing transformation. In *Proceedings of the 21st annual conference on computer graphics and interactive techniques* (pp. 451–458). New York, NY, USA: ACM. Retrieved from `http://doi.acm.org/10.1145/192161.192283` doi: 10.1145/192161.192283

Larsen, M., Labasan, S., Navrátil, P., Meredith, J. & Childs, H. (2015, May). Volume Rendering Via Data-Parallel Primitives. In *Proceedings of eurographics symposium on parallel graphics and visualization (egpgv)* (p. 53-62). Cagliari, Italy.

Larsen, M., Meredith, J. S., Navrtil, P. A. & Childs, H. (2015, April). Ray tracing within a data parallel framework. In *2015 ieee pacific visualization symposium (pacificvis)* (p. 279-286). doi: 10.1109/PACIFICVIS.2015.7156388

Lessley, B., Binyahib, R., Maynard, R. & Childs, H. (2016, June). External Facelist Calculation with Data-Parallel Primitives. In *Proceedings of eurographics symposium on parallel graphics and visualization (egpgv)* (p. 10-20). Groningen, The Netherlands.

Lessley, B., Moreland, K., Larsen, M. & Childs, H. (2017, October). Techniques for Data-Parallel Searching for Duplicate Elements. In *Proceedings of ieee symposium on large data analysis and visualization (ldav)* (pp. 1–5). Phoenix, AZ.

Lessley, B., Perciano, T., Mathai, M., Childs, H. & Bethel, E. W. (2017, October). Maximal Clique Enumeration with Data-Parallel Primitives. In *Proceedings of ieee symposium on large data analysis and visualization (ldav)* (pp. 16–25). Phoenix, AZ.

Levoy, M. (1990, July). Efficient ray tracing of volume data. *ACM Trans. Graph.*, *9*(3), 245–261. Retrieved from `http://doi.acm.org/10.1145/78964.78965` doi: 10.1145/78964.78965

Li, S., Marsaglia, N., Chen, V., Sewell, C., Clyne, J. & Childs, H. (2017, June). Achieving Portable Performance For Wavelet Compression Using Data Parallel Primitives. In *Proceedings of eurographics symposium on parallel graphics and visualization (egpgv)* (pp. 73–81). Barcelona, Spain.

Lindemann, F. & Ropinski, T. (2011, Dec). About the influence of illumination models on image comprehension in direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, *17*(12), 1922-1931. doi: 10.1109/TVCG.2011.161

Lo, L.-t., Sewell, C. & Ahrens, J. P. (2012). Piston: A portable cross-platform framework for data-parallel visualization operators. In *Egpgv* (pp. 11–20).

Lokovic, T. & Veach, E. (2000). Deep shadow maps. In *Proceedings of the 27th annual conference on computer graphics and interactive techniques* (pp. 385–392). New York, NY, USA: ACM Press/Addison-Wesley Publishing Co. Retrieved from `http://dx.doi.org/10.1145/344779.344958` doi: 10.1145/344779.344958

Max, N. (1995, Jun). Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, *1*(2), 99-108. doi: 10.1109/2945.468400

Maynard, R., Moreland, K., Atyachit, U., Geveci, B. & Ma, K.-L. (2013). Optimizing threshold for extreme scale analysis. In *Is&t/spie electronic imaging* (pp. 86540Y–86540Y).

Moreland, K., Sewell, C., Usher, W., Lo, L., Meredith, J., Pugmire, D., ... Geveci, B. (2016, May/June). VTK-m: Accelerating the Visualization Toolkit for Massively Threaded Architectures. *IEEE Computer Graphics and Applications (CG&A)*, *36*(3), 48-58.

Mueller, K., Moller, T. & Crawfis, R. (1999, Oct). Splatting without the blur. In *Visualization '99. proceedings* (p. 363-544). doi: 10.1109/VISUAL.1999.809909

Newman, M. & Barkema, G. (1999). *Monte carlo methods in statistical physics.* Oxford University Press: New York, USA.

Parker, S., Parker, M., Livnat, Y., Sloan, P.-P., Hansen, C. & Shirley, P. (2005). Interactive ray tracing for volume visualization. In *Acm siggraph 2005 courses.* New York, NY, USA: ACM. Retrieved from `http://doi.acm.org/10.1145/1198555.1198754` doi: 10.1145/1198555.1198754

Parker, S., Shirley, P., Livnat, Y., Hansen, C. & Sloan, P.-P. (1998). Interactive ray tracing for isosurface rendering. In *Proceedings of the conference on visualization '98* (pp. 233–238). Los Alamitos, CA, USA: IEEE Computer Society Press. Retrieved from `http://dl.acm.org/citation.cfm?id=288216.288266`

Parsonson, L., Bai, L., Bourn, L., Bajwa, A. & Grimm, S. (2011). Medical imaging in a cloud computing environment. In *CLOSER* (pp. 327–332).

Raab, M., Seibert, D. & Keller, A. (2008). Unbiased global illumination with participating media. In *Monte carlo and quasi-monte carlo methods 2006* (pp. 591–605). Springer.

Ropinski, T., Dring, C. & Rezk-Salama, C. (2010, March). Interactive volumetric lighting simulating scattering and shadowing. In *2010 ieee pacific visualization symposium (pacificvis)* (p. 169-176). doi: 10.1109/PACIFICVIS.2010.5429594

Rushmeier, H. (1988). *Realistic image synthesis for scenes with radiatively participating media.* Cornell University, May. Retrieved from `https://books.google.com/books?id=EJdInQEACAAJ`

Schlegel, P., Makhinya, M. & Pajarola, R. (2011). Extinction-based shading and illumination in gpu volume ray-casting. *IEEE Transactions on Visualization and Computer Graphics*, *17*(12), 1795–1802.

Schroots, H. A. & Ma, K.-L. (2015). Volume Rendering with Data Parallel Visualization Frameworks for Emerging High Performance Computing Architectures. In *Siggraph asia 2015 visualization in high performance computing* (pp. 3:1–3:4). ACM.

Wanger, L. R., Ferwerda, J. A. & Greenberg, D. P. (1992, May). Perceiving spatial relationships in computer-generated images. *IEEE Computer Graphics and Applications*, *12*(3), 44-58. doi: 10.1109/38.135913

Westover, L. (1990). Footprint evaluation for volume rendering. In *Proceedings of the 17th annual conference on computer graphics and interactive techniques* (pp. 367–376). New York, NY, USA: ACM. Retrieved from `http://doi.acm.org/10.1145/97879.97919` doi: 10.1145/97879.97919

Widanagamaachchi, W., Bremer, P.-T., Sewell, C., Lo, L.-T., Ahrens, J. & Pascuccik, V. (2014). Data-parallel halo finding with variable linking lengths. In *Large data analysis and visualization (ldav), 2014 ieee 4th symposium on* (pp. 27–34).

Zhukov, S., Iones, A. & Kronin, G. (1998). An ambient light illumination model. In G. Drettakis & N. Max (Eds.), *Rendering techniques '98* (pp. 45–55). Vienna: Springer Vienna.